8 prerequisites of a graph query language

Mingxi Wu
WHO AM I?

Mingxi Wu

- Ph.D. in Database & Data Mining, University of Florida 2008
- SDE SQL server group, Microsoft 2007
- SDE relational database optimizer group, Oracle 2008-2011
- Lead SDE big data management group, Turn Inc. 2011-2014
- VP Engineering, TigerGraph 2014- now
Why Graph?

Graph Model Is Advantageous

- To unleash the power of interconnected data for deeper insights and better outcomes
- Intuitive and clear data model and visual representation
- Other DBs can’t traverse multiple links like a Native Graph DB can
Why A Graph Language?

- Graph Guru is hard to train and find on market
- No standard language slow down enterprise adoption
- A high declarative language lower the barrier to the gap
8 Prerequisite Of A Graph Language

- Schema based with capability of schema evolvement
- High-level control of graph traversal—pattern matching
- Fine control of graph traversal—accumulator
- Built-in parallel semantic to ensure high performance
- A highly expressive loading language - basic tranfromation
- Data Security and Privacy—multiple graph + RBAC
- Support Query Composability—stored procedure
- SQL user friendly
1 - Schema Based With Evolvement

- Data independency
  - Data independent application dev
  - Separate meta data and binary, high compression
- Schema evolvement
  - Needed in real-life cases
  - Agile for business grow adaption
2 - High level Control of Graph Traversal

- Declarative abstract away of how to crunching data
  - Pattern match
- Stay in high level is more productive and easy to maintain
3 - Fine Control Of Graph Traversal

- Large application rely on coding iterative algorithm with customized logic—**need accumulator** and **flow control**
  - PageRank
  - Community Detection
  - Centrality
  - Complexed application logic
4 - Built-in Parallel Semantic To Ensure Performance

- Graph algorithm is expensive
  - Each hop exponentially add more data
- Built-in parallel semantic help performance and thinking
PARALLEL ILLUSTRATION
PARALLEL ILLUSTRATION
PARALLEL ILLUSTRATION
PARALLEL ILLUSTRATION

Start node

Node 1

Node 2

Node 3

Node 4

Node 5

Node 6

Node 7

Node 8

Node 9

End node
PARALLEL ILLUSTRATION
PARALLEL ILLUSTRATION
5 - Highly Expressive Loading Language

- World is a graph
- Ingesting data silos and handle heterogeneity need
  - expressive & flexible mapping support
  - Customized token transformations
- #1 criteria to evaluate a high quality graph db
6 - Data Security and Privacy

- Enterprise user keen on collaboration on data
  - Collaboration
  - Meanwhile, privacy
- Solution
  - Multiple Graph — Sharing + Privacy
  - Role-based access control (RBAC)
Batch Query need
- E.g. want to recommend for a set of users
- Same algorithm for each user
- A for-loop + a stored procedure
- Divid-and-conquer reduce graph algorithm complexity
8 - SQL User Friendly

- Graph Query and Application is new
- SQL user base is “stubborn" and mass
- Shorten the gap between SQL and Graph Language
  - Speedup adoption
  - Smooth transition
What’s out there on the Market?

- Gremlin - functional chain style, Turing complete
- Cypher - Pattern match style, SQL complete
- Sparql - Pattern match and more SQL style, SQL complete
- GSQL - Pattern match + accumulator + flow control, Turing complete
Gremlin - functional language, Turing complete

Language Model
- Property Graph G + Traversal Tao + Set of Traversers T
- Result: the halted Traversers’ locations.

Traversal style: `g.V().hasId("2").outE().inV()`

Match style:
- `g.V().match(as("a").out("teach").as("b"), as("a").out("registered").as("c")).dedup(a).select("a").by("name")`

Branching:
- `g.V().hasLabel('stock').choose(values('ticker')).option('AMZN', values('price')).option('FB', values('30Day-Avg'))`

Runtime Attribute flow: each traverser carry a “sack”, local variable
Gremlin - Pros and Cons

**Pros**
- Expressive - Turing complete
- Apache interactive shell - easy to start

**Cons**
- Thinking complexity is high - exponential runtime tree
- Hard to do simple runtime computation when multiple passes is needed
- Not SQL user-friendly
- Query Calling Query is not native syntax
- No flexible loading language
Simple Question: $\text{sum}(v5+v6) - \text{sum}(v3+v4)$
Simple Question: $\text{sum}(v5+v6)-\text{sum}(v3+v4)$

```
g.V(2).union(outE().has('weight',1).inV().sack(assign).by('vvalue').sack(mult).
by(constant(-1)).sack().sum(), outE().has('weight', 2).inV().values('vvalue').sum()).sum()
```
Cypher - Neo4j, early 2011-

- Cypher - declarative, pattern match, SQL-complete
- Language Model
  - Property Graph G + sequential or composition of Table functions
  - Result: table output
- Match style:
  - `MATCH (a:teacher)-[r:teach]-(b:subject)
    RETURN a.name, count(distinct b) as subjCnt`
- Tuple Flow style:
  - `MATCH (a:teacher) -[r:teach]-> (b:subject)
    WITH a, count(distinct b) as subjCnt
    MATCH (a) -[t:has_title]-> (c:title)
    RETURN a.name, subjCnt, c.title_name`
- Branching:
  - Very limited, if-then-else, loop is hard.
- Runtime Attribute flow: just as in SQL, augment output and flow to next table function
Cypher - Pros and Cons

Pros
- Easy for relational-mind transition to graph
- Borrow many from SQL (WHERE, GROUP BY, ORDER BY)

Cons
- Not too expressive for graph - SQL complete
- Flow control support very limited
- Query composability is not in native syntax
- Data dependent
- Iterative algorithm of graph (hard)
Simple Question: \( \text{sum}(v5+v6) - \text{sum}(v3+v4) \)
Simple Question: \(\text{sum}(v5+v6)-\text{sum}(v3+v4)\)

MATCH a:V - [e:E]- b:V
WHERE a.id = "v2" AND e.weight = 2
WITH a, SUM(b.value) as sum1
MATCH (a) - [e:E]- d:V
RETURN a, sum1 - SUM(d.value)
Sparql - declarative, triplet pattern match, SQL-complete

Language Model
- RDF Graph G + conjunction/disjunction of triplet table functions
- Result: table output

Match style:
- `PREFIX foaf: <http://xmlns.com/foaf/0.1/>`
  
  
  `SELECT ?name ?email`
  
  `WHERE { ?person a foaf:Person .`
  
  `?person foaf:name ?name .`
  
  `?person foaf:mbox ?email .}`

Branching:
- Very limited, if-then-else, loop is hard.
- Runtime Attribute flow: just as in SQL, create graph view or use subquery
Sparql- Pros and Cons

Pros
- Easy for RDF characteristic
- Borrow many from SQL (WHERE, GROUP BY, ORDER BY)

Cons
- Not too expressive - SQL complete
- Flow control support very limited
- Query Composability is not in native syntax
- Not for property graph
- Fine control of graph (hard)
- GSQL - declarative, PL/SQL style or Stored Procedure style
- GSQL - turing complete
- Language Model
  - Property Graph G + DAG of GSQL query blocks
  - Result: graph or table format
- Language style:
  - composed by many **single SQL** block
- Branching:
  - If-then-else, While, Foreach
- Runtime Attribute flow: accumulator attached to vertices, complexity is O(V).
Simple Question: $\text{sum}(v5+v6)-\text{sum}(v3+v4)$
Start = \{v2\};

Result = SELECT v
FROM Start-(:e)->:tgt
ACCUM
    CASE WHEN e.w == 1 THEN
    Start.@sum1 += tgt.val;
    CASE WHEN e.w == 2 THEN
    Start.@sum2 += tgt.val;
    END;
    POST-ACCUM @@result = Start.@sum2 - Start.@sum1;

PRINT @@result;
GSQIL loading language

**GSQIL commands to define a loading job**

```
USE GRAPH social
BEGIN
CREATE LOADING JOB load_social FOR GRAPH social {
   DEFINE FILENAME file1="/home/tigergraph/person.csv";
   DEFINE FILENAME file2="/home/tigergraph/friendship.csv";

   LOAD file1 TO VERTEX person VALUES ("name", "name", "age", "gender", "state") USING header="true", separator="";
   LOAD file2 TO EDGE friendship VALUES (0, 1, 2) USING header="true", separator="";
}
END
```
GSQ L - Pros and Cons

Pros

▪ Expressive - Turing complete
▪ Flow control support
▪ Query Composability is in native syntax
▪ Fine control of graph with accumulators
▪ Expressive and elegant loading language

Cons

▪ Less seen by graph community, but getting more and more popular
Path Legality Semantics: $1- [E^*] - 5$

- Infinite number of paths (**Gremlin**)
- Three non-repeated-vertex paths (1-2-3-4-5, 1-2-6-4-5, and 1-2-9-10-11-12-4-5)
- Four non-repeated-edge paths (1-2-3-4-5, 1-2-6-4-5, 1-2-9-10-11-12-4-5, and 1-2-3-7-8-3-4-5); (**Cypher**)
- Two shortest paths (1-2-3-4-5 and 1-2-6-4-5) (**GSQL**)

![Diagram of a graph with nodes labeled 1 to 12 and edges connecting them. The paths mentioned in the text are highlighted on the diagram.]
1-Hop Atomic Pattern

- 1-hop pattern
  - FROM X:x - (E1:e1) - Y:y
    - Undirected edge
  - FROM X:x - (E2>e2) - Y:y
    - Right directed edge
  - FROM X:x - (<E3:e3) - Y:y
    - Left directed edge
  - FROM X:x - (:e) - Y:y
    - Any undirected edge
  - FROM X:x - (>:e) - Y:y
    - Any right directed
  - FROM X:x - (<:e) - Y:y
    - Any left directed
  - FROM X:x - (<[:e]) - Y:y
    - Any left directed and any undirected
  - FROM X:x - (E1|E2>|<E3):e - Y:y
    - Disjunctive 1-hop edge
  - FROM X:x - () - Y:y
    - any edge (directed or undirected) match this 1-hop pattern
      - [<_>]<_>
  - Syntax sugar
    - FROM X:x - ((E1|E2>|<E3):e) - Y:y
1-Hop Star Pattern

- 1-hop star pattern — repetition of an edge pattern, 0 or more
  - FROM X:x - (E1*) - Y:y
  - FROM X:x - (E2>*) - Y:y
  - FROM X:x - (<E3*) - Y:y
  - FROM X:x - (_) - Y:y
  - FROM X:x - ((E1|E2>|<E3)*) - Y:y
    - Cypher does not have this
  - FROM X:x - ((E1.<E3*) - Y:y
    FROM X:x - ((-E1-.<-E3-)*) - Y:y
    - Repeat a path with “*”, not supported; Neither Cypher.
  - No alias binding allowed for variable length pattern
1-Hop Star Pattern With Bounds - Shortest Path Match Only.

- 1-hop star pattern with bounds
  - FROM X:x - (E1*2..) - Y:y
    - **Lower bounds only.** At least 2.
  - FROM X:x - (E2>*..3) - Y:y
    - **Upper bounds only.** 0 up to 3.
  - FROM X:x - (<E3*3..5) - Y:y
    - **Both Lower and Upper bounds.** 3 to 5 repetitions
  - FROM X:x - ((E1|E2>|<E3)*3) - Y:y
    - **Exact bound.** exactly 3 repetitions.
Multiple Hop Pattern—Succinct Representation

- 2-hop pattern
  - FROM X:x-(E1:e1)-Y:y-(E2>:e2)-Z:z
  - FROM X:x-\((E1.E2)\)-Y:y
    - Concatenation of Edges with “.” unlimited in countable set
    - E1.E2.E3.E4…………
    - No alias binding of the concatenated edge pattern.

- 3-hop pattern
  - No alias binding allowed for path pattern
Circular Pattern

- FROM X:x-(E*1..)-Y:y WHERE x == y
- FROM X:x-(E1>.<E2.E4*)-Y:y WHERE x == y
CREATE QUERY ic_6(VERTEX<Person> personId, STRING tagName) FOR GRAPH ldbc_snb {
    TYPEDEF tuple<STRING tagName, INT postCount> tagStats;
    SetAccum<VERTEX<Post>> @@postAll;
    SumAccum<INT> @postCount;
    HeapAccum<tagStats>(10, postCount DESC, tagName ASC) @@tagStatsTop;
    vPerson = { personId };
    aggPersonPostTag =
        SELECT t3
        FROM vPerson:s
        -((Person_KNOWS_Person|<Person_KNOWS_Person>*1..2)-Person:t1
        -(<Post_HAS_CREATOR_Person:e2>-Post:t2
        -<Post_HAS_TAG_Tag>:e3)-Tag:t3
        WHERE s != t1
        ACCUM CASE WHEN t3.name == tagName THEN @@postAll += t2 END;
    vPost = { @@postAll };
    vTag =
        SELECT t
        FROM vPost:s-(Post_HAS_TAG_Tag>:e)-Tag:t
        ACCUM CASE WHEN t.name != tagName THEN t.@postCount += 1 END
        POST-ACCUM @@tagStatsTop += tagStats(t.name, t.@postCount);
    PRINT @@tagStatsTop;
}
GSQ Developer Resource

- Developer portal https://www.tigergraph.com/developers/
- Developer Forum https://groups.google.com/a/opengsql.org/forum/#!forum/gsql-users (our dev do free support on language there)
- Open sourced graph library in GSQL https://docs.tigergraph.com/graph-algorithm-library
TigerGraph Recent Awards
Customers

Selected Paid Customers

- Intuit
- Uber
- Alipay
- Wish
- Zillow
- Department of Santa Clara County
- China Merchants Bank
- Ice Kredit
- China Mobile
- VISA
- Large US Based Pharma
- State Grid Corporation of China
MaxAccum<float> @@maxDifference = 9999;  # max score change in an iteration
SumAccum<float> @received_score = 0;  # sum of scores each vertex receives from neighbors
SumAccum<float> @score = 1;  # initial score for every vertex is 1.

AllV = {Page.*};  # Start with all vertices of type Page
WHILE @@maxDifference > maxChange LIMIT maxIteration DO
    @@maxDifference = 0;
    S = SELECT s
        FROM AllV:s-(Linkto)->:t
        ACCUM t.@received_score += s.@score/s.outdegree()
        POST-ACCUM s.@score = dampingFactor + (1-dampingFactor) * s.@received_score,
                        s.@received_score = 0,
                        @@maxDifference += abs(s.@score - s.@score');
    PRINT @@maxDifference;  # print to default json result
END;  # end while loop
#PRINT AllV.page_id, AllV.@score;  # print the results, JSON output API version v1
PRINT AllV[AllV.page_id, AllV.@score];  # print the results, JSON output API version v2
USE GRAPH social
CREATE QUERY hello2 (VERTEX<person> p) FOR GRAPH social{
    OrAccum @visited = false;
    AvgAccum @@avgAge;
    Start = {p};

    FirstNeighbors = SELECT tgt
                     FROM Start:s -(friendship:e)-> person:tgt
                     ACCUM tgt.@visited += true, s.@visited += true;

    SecondNeighbors = SELECT tgt
                      FROM FirstNeighbors -(e)-> :tgt
                      WHERE tgt.@visited == false
                      POST_ACCUM @@avgAge += tgt.age;

    PRINT SecondNeighbors;
    PRINT @@avgAge;
}
INSTALL QUERY hello2
RUN QUERY hello2("Tom")